Optimized Tile-Based Texture Synthesis

Weiming Dong
Project ALICE, INRIA Lorraine/LORIA
BP 239 - 54506 Vandoeuvre-lès- Nancy cedex, France
Weiming.Dong@loria.fr

Ning Zhou
Tsinghua University
100084 Beijing, P.R.China
zhoun03@mails.tsinghua.edu.cn

Jean-Claude Paul
INRIA/Tsinghua University
100084 Beijing, P.R.China
paul@tsinghua.edu.cn

ABSTRACT

One significant problem in tile-based texture synthesis is the presence of conspicuous seams in the tiles. The reason is that the sample patches employed as primary patterns of the tile set may not be well stitched if carelessly picked. In this paper, we introduce an optimized approach that can stably generate an ω-tile set of high pattern diversity and high quality. Firstly, an extendable rule is introduced to increase the number of sample patches to vary the patterns in an ω-tile set. Secondly, in contrast to the other concurrent techniques that randomly choose sample patches for tile construction, our technique uses Genetic Algorithm to select the feasible patches from the input example. This operation insures the quality of the whole tile set. Experimental results verify the high quality and efficiency of the proposed algorithm.

Keywords: Texture synthesis, $\omega - tile$, Genetic Algorithm (GA), sample patches selection.

1 INTRODUCTION

Generating novel photo-realistic imagery from smaller examples has been widely recognized as significant in computer graphics. A wide number of applications require realistic textures to be synthesized for object decoration in virtual scenes. The global repeatability within texture images is essential to texture synthesis techniques. This inherent property makes it possible to express adequate texture information with limited portions.

Nowadays local region-growing methods are popularly used in texture synthesis. These methods generate the texture by growing one pixel [3, 8, 28, 2] or patch [7, 14, 20, 29, 19, 13] at a time with the constraint of maintaining coherence of neighboring pixels in the grown region. Such approaches always suffer the time-consuming neighborhood matching in the example and do not sufficiently meet real-time applications. On the other hand, some near real-time texture synthesis methods usually achieved low quality results for the lack of optimization in the pre-processing [30, 31] or needed very complex pre-computation and data structures [17]. Recently efficient GPU-based texture synthesis techniques [15, 16] have also been proposed, however they always demand a high performance graphics hardware.

An alternative approach is to pre-compute a set of small tiles and use these tiles to generate arbitrary size of non-periodic images at run time [4, 27, 21, 6]. The tile-based method usually employs a set of *sample patches* which are extracted from the input example as texturing primitive. Then tiles are constructed by stitching sample patches together following some given rules. This technique requires only a small amount of memory and is very useful in many real-time applications, although sometimes achieving low quality results or dull patterns for the lack of optimization on the tile set.

In this paper, we present an approach for tile-based texture synthesis that is based on the optimization of tile set quality within a *Genetic Algorithm* (GA)-based framework. Our main contribution is to merge some locally defined optimization measures into a global *evaluation function* that can jointly optimize the quality of the entire tile set. This evaluation function balances the qualities among tiles and can be optimized by GA with reasonable computational cost. GA is an efficient and near global optimum search method [25, 12, 18, 23], which can automatically achieve and accumulate knowledge about the search space and adaptively control the search process to approach the global optimal solution.

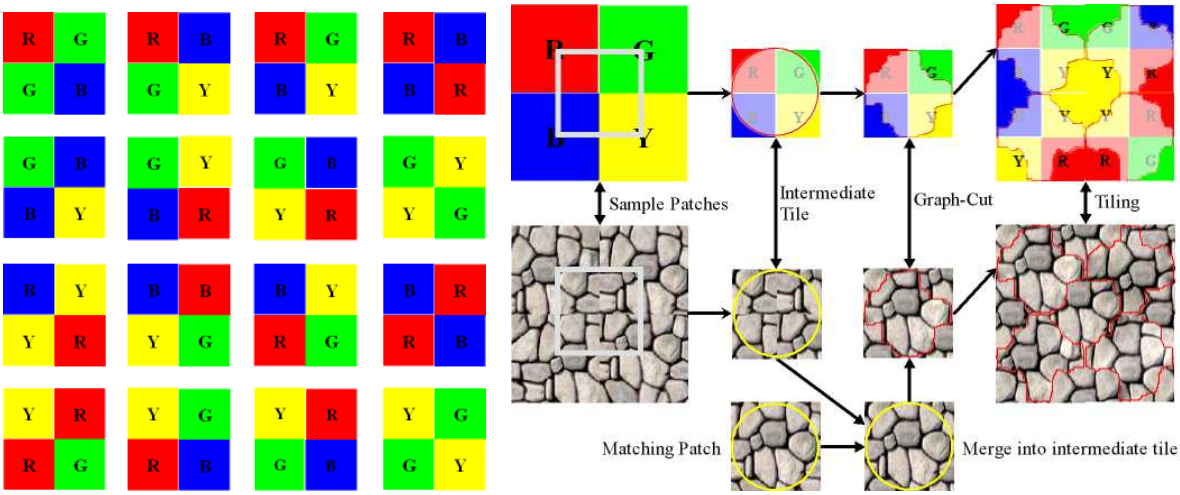
Our approach starts with an 8-tiles ω-tile set used in [21] as the basic tile set (Figure 2(a)). An extendable rule will be given to directly derive new tile sets from existing ones. By employing more sample patches during the tile construction process, the new ω-tile sets introduced in this paper can generate patterns more abundant than the sets of similar sizes in both [21] and [4].

As shown in Figure 1(b), an ω-tile is constructed by finding a matching patch from the input, and merging it into the intermediate tile using graph-cut. The *intermediate tile* is the combination of four sample patch quarters. We propose a global optimization algorithm to search for a feasible set of sample patches. Intermediate tiles formed by these patches will satisfy both local and global optimal conditions. The local condition implies that each intermediate tile could find an adequately well *matching patch* (the texture patch picked from the input example to merge into the intermediate tile for erasing junctions) from the input, while the global condition means that the tile qualities are balanced according to their smallest matching errors. The two conditions are interpreted together as an *evaluation function*. This function is defined as the linear combination of the sum of the *matching errors* (the distance between the intermediate tile and the candidate matching patch) between intermediate tiles and their *closest matching patches* (the candidate matching patch with the smallest distance), and the standard variance of all the errors. Sample patches selection proceeds by optimizing this evaluation function using GA. We start searching with a group of sample patches extractions. The optimization procedure improves the quality of the entire group through successive iterations of the algorithm. The final solution is the best extraction in the group when GA terminates.

2 RELATED WORK

Texture tiling. Cohen et. al [4] developed a stochastic algorithm to non-periodically tile the plane with a small set of Wang-tiles at run time. Wei [27] extended this work with GPU to improve tile-based texture mapping. Ng et al. [21] presented another approach to generate a set of small texture tiles from an input example. These tiles could also be tiled together to synthesize large textures. Our technique uses their ω-tiles as the tile set pattern. Figure 1(a) shows a typical ω-tile set in [21]. All these approaches require a set of sample patches extracted from the input example to generate the primary tile patterns, so the quality of their texture tiling results is not stable due to the uncertainty of the sample patches.

The previous ω-tile construction process is shown in Figure 1(b).



(a) Set of 16 ω-tiles in [21].

(b) Basic construction process of an ω-tile. The matching patch is merged into the intermediate tile by graph-cut.

Figure 1: The previous ω-tile set formation process.

The approach starts with randomly obtaining a set $\mathbb{T}$ of square sample patches from the input example. Each patch is assigned to be related with one color square. Then an intermediate tile can be cut from the middle of the texture block. Different intermediate tiles are generated according to the different arrangements of the sample patches (corresponding to the arrangements of the color squares). The cross junctions where four quarters meet is erased by picking a matching patch from the input and merging it into the intermediate tile with graph-cut. The matching patch is the same size as the intermediate tile. A circle is employed to constrain the boundary of the cutting curve in order to maintain the continuity of the patterns between matching sides in the tiling image.

Genetic Algorithm. GA is a stochastic search method for solving optimization problems. It is so named because the scheme is based on the mechanics of natural selection and natural genetics. Research interests in heuristic search algorithms with underpinnings in natural and physical processes began in the 1970s, when Holland [10] proposed GA. GA generates a sequence of populations using a selection mechanism, and applies crossover and mutation as search mechanisms. It has demonstrated considerable success in providing good solutions to many complex optimization problems [18, 26], such as capital budgeting, vehicle routing problem, critical path problem, parallel machine scheduling, redundancy optimization, open inventory network etc. The advantage of GA is due to its ability to obtain a global optimal solution fairly in a multidimensional search landscape, which has several locally optimal solutions as well. Hence it is a powerful technique that can be applied in texture synthesis.

3 Tile Set Formation

We choose the size-8 ω-tile set in [21] as the basic tile set, as shown in Figure 2(a). Here we use $\mathbb{B}$ to denote it. A set of squares $\mathbb{P} = \{R, G, B, Y\}$ are used to compose the color blocks.

3.1 Tile Patterns Analysis

The tiling process using ω-tiles is carried out in scan-line order. Once the tile in the left-top corner is fixed, the rest tiles in the tiling are laid one by one from left to right, and top to bottom, consistent with the square colors of their neighbors. A valid tiling using $\mathbb{B}$ is shown in Figure 2(b). The basic tile set composed of only 8 tiles will draw undesirable repetitive patterns in a large synthesis image. Two methods are proposed in [21] to overcome this artifact. One way is to pick two patches from the input for each original tile. It can only partly eliminate the repetitive patterns because we should not neglect the repetitive patterns caused by the central parts of the sample patches. As shown in the rightmost column of Figure 1(b), the central pattern of the sample patch still possesses an important role in the tiling. To solve this problem, we develop an effective method to directly increase the number of the sample patches, without losing the characteristics of the whole tile set. The other way used in [21] is to increase the tiles quantity by using more arrangements of the sample patches. This method is also suitable for our approach.

3.2 Increase Sample Patches

We add new patterns into the ω-tile set by enlarging the size of the sample patches set $\mathbb{T}$. This operation is proceeded directly on $\mathbb{B}$. We derive new tiles with the following steps:

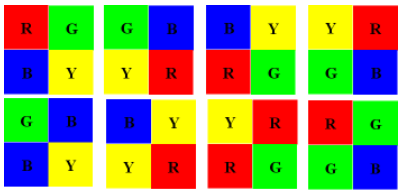
1. Randomly pick a square from set $\mathbb{P}$ as the "reference" square (or reference color). Without losing generality, we choose the yellow square as the reference square.

2. Add a new square to the set $\mathbb{P}$. Here we use C (Cyan) to represent it. Then the new square set is enlarged to be $\mathbb{P}_1 = \{R, G, B, Y, C\}$.

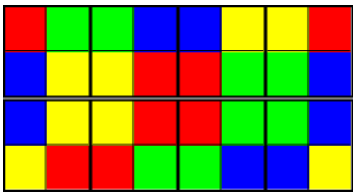
3. Generate new tiles by replacing the yellow squares with cyan squares in $\mathbb{B}$.

4. Add another two tiles by replacing only one yellow square with cyan in the tile $\langle B, Y, Y, R\rangle$.

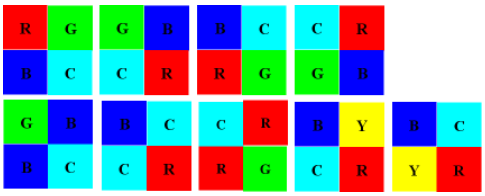
The new tiles formed by the above method is shown in Figure 2(c). The last two tiles are the additional tiles generated by step 4. The new tiles together with $\mathbb{B}$ forms the new ω-tile set $\mathbb{N}_1$. Despite of the two "additional" tiles, the other new tiles plus the tile $\langle R, G, G, B\rangle$ can be considered as a copy of the basic set $\mathbb{B}$. It can be used independently to tile arbitrary size area. We use $\mathbb{N}_1'$ to



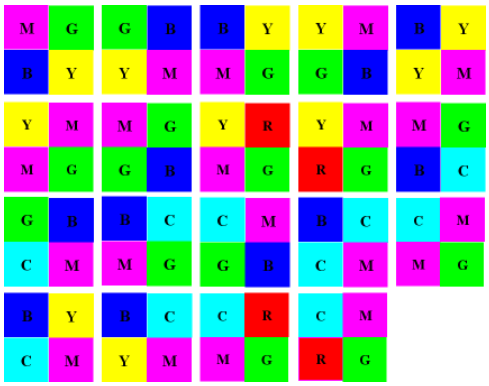
(a) Basic ω-tile set



(b) A valid tiling using the tile set in (a)



(c) Add one color to the basic set

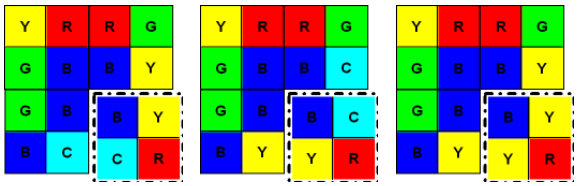


(d) Add the second color to the basic set

Figure 2: Tile set formation from basic size-8 ω-tile set

indicate the tile set which possesses all the tiles in $\mathbb{N}_1$ except the two additional tiles. During the tiling process, if only $\mathbb{N}_1'$ is used, there will be no problem if no yellow square and cyan square need to be matched at the same time when placing a tile. It can be treated as a pattern duplication with the basic tile set. The cases shown in Figure 3(a) and Figure 3(b) appear where yellow square and cyan squares are needed to be matched simultaneously, there will be no matching tile in $\mathbb{N}_1'$. Now the two additional tiles are necessary to complete the tiling process. In fact these two patterns are just the extensions of the pattern in Figure 3(c), which is one of the tiling patterns using the basic set $\mathbb{B}$. Therefore, the new tile set $\mathbb{N}_1$ can be used to tile any large area and maintains all the properties of the ω-tile set which are stated in [21].

We can continue to increase the number of sample patches into six by following the same rule. Based on the tile set $\mathbb{N}_1$, we pick red as the reference color, and generate new tiles by replacing the red squares with magenta. The tiles with double red squares also need to be derived into two additional tiles by replacing only one square at a time. The new tiles are shown in Figure 2(d). Packed



(a) (b) (c)

Figure 3: The cases why the two additional tiles in Figure 2(c) are required.

with set $\mathbb{N}_1$, we get another new ω-tile set $\mathbb{N}_2$. Here the square set is enlarged to $\mathbb{P}_2 = \{R, G, B, Y, C, M\}$. Obviously, we can keep on increasing the sample patches in $\mathbb{P}_2$ by following this rule if there is enough memory during the rendering process. Thus in the tiling process, this method can effectively eliminate the repetitive patterns caused by both the tiles and the sample patches themselves. Usually the set $\mathbb{N}_2$ which contains 38 tiles is enough to achieve a less-repetitive tiling result.

4 Tile Construction

The tiling quality is directly decided by the quality of the final tile set. As illustrated in Figure 1(b), the graph-cut result performed between the intermediate tile and the matching patch will directly affect the tile quality. Thus, special care should be taken when picking sample patches from the input, so that the intermediate tiles can get feasible matching patches for graph-cut operations. The random picking method used in [4] and [21] is not optimal for this requirement. The intermediate tile formed by randomly chosen sample patches can not insure the finding of a good matching patch from the example. On the other hand, because of the great quantity of pixels in an image, it is almost impossible to use brute-force searching method to find the best sample patches extraction. For example, for an input example of size 128×128, if we set the tile size to be 80×80, there will be $(48 \times 48)^n$ choices for the sample patches extraction (n is the number of sample patches). It is an unacceptable computational requirement for a normal PC in reasonable time. Our approach can efficiently and accurately solve this problem.

4.1 Algorithm Overview

Our optimized sample patch selection algorithm is essentially based on GA. GA starts with an initial set of randomly generated chromosomes called a population where each chromosome encodes a solution to the optimization problem. All chromosomes are evaluated by an evaluation function which is some measure of fitness. A selection process based on the fitness values will form a new population. The cycle from one population to the next is called a generation. In each new generation, all chromosomes will be updated by the crossover and mutation operations. Then the selection process selects chromosomes to form a new population. After performing a given number of cycles, or other termination criteria is satisfied, we denote the best chromosome into a solution, which is regarded as the optimal solution of the optimization problem. We will follow this framework to develop our algorithm.

4.2 Optimized Sample Patches Selection

The original problem of sample patches selection is to search for n sample patches from the input example to fill the m tiles. For example for the ω-tile set $\mathbb{N}_2$, $\{n = 6, m = 38\}$. Then the intermediate tiles filled by the these sample patches can safely find

feasible matching patches under the given rules for junction flattening. Here we use GA to optimally find the n sample patches:

Initialization: To ensure an optimal solution be obtained in a reasonable runtime, an initial population consists of a considerable amount of chromosomes (*n* sample patches) is necessary. To start the algorithm, an integer *pop_size* is defined as the number of chromosomes. From the input example, *pop_size* chromosomes is selected randomly, denoted by $\{A_1, A_2, ..., A_{pop\_size}\}$. Every chromosome contains *n* sample patches (*n* genes) selected from the example image:

$$A_i = (g_i^1, g_i^2, ...g_i^n), i = 1, 2, ..., pop\_size$$

We encode each gene as the coordinate of the sample patch in the input example.

Evaluation: In GA, the selection of chromosomes to reproduce is determined by a probability assigned to each chromosome A_i. This probability is proportional to its fitness relative to other chromosomes in the population, i.e. chromosomes with higher fitness will have more chance to produce offsprings by the selection process. In the context of sample patches selection, the fitness of a chromosome, is evaluated by an evaluation function, $E(A_i)$ which measures the performance of the sample patches derived from that chromosome. The evaluation function, in essence, computes the minimum matching error between the intermediate tile and the candidate matching patch. It involves searching for translations of the input image that match well with the intermediate tile. Let $I(p)$ and $T(p)$ be the pixels at the position p in the input example and the intermediate tile, the evaluation function is defined as:

$$C_j(t) = \sum_{p \in S_t} |T(p) - I(p-t)|^2, (j = 1, 2, ..., m; t \in P_T) \quad (1)$$

$$D_j = \min \{C_j(t), t \in P_T\}$$
$$V_i = Variance(D_1, D_2, ..., D_m)$$
$$E(A_i) = \lambda \cdot \sum_{j=1}^{m} D_j + (1 - \lambda) \cdot V_i$$

where $C_j(t)$ is the matching error at translation t of the jth tile. It defines the distance between the intermediate tile and the candidate matching patch. S_t is the portion of the translated input overlapping the tile, P_T is the set of valid translations (candidate matching patches) in the input, and D_j is the minimum matching error within all the translations. V_i is the variance of all the minimum errors, we add this factor in order to protect the global quality of the final tile set. It avoids that intermediate tiles with extremely high and extremely low matching errors appear together in the same set. So the evaluation function $E(A_i)$ is the linear combination of the minimum error sum and the variance. We set $\lambda = 0.8$ for all the experiments in this paper. Note that our evaluation function is very similar to the energy function used in [13] for texture optimization, while here we use GA rather than Expectation Maximization (EM) to optimize it.

To obtain an optimal tile set, we need to determine the best sample patches that has the least overall matching error. Hence, the fitness function $F(A_i)$ for selection is defined as the inverse of the evaluation function, i.e. $F(A_i) = \frac{1}{E(A_i)}$. Based on the value of the fitness function for each chromosome, the population of chromosomes $\{A_1, A_2, ..., A_{pop\_size}\}$ is rearranged from high fitness to low fitness.

Selection: The selection process is basically a "spinning the roulette wheel" process. The roulette wheel is spun *pop_size* times and each time a chromosome from the rearranged population $\{A_1, A_2, ..., A_{pop\_size}\}$ is selected. As we have stated, the chromosome with higher fitness should have a higher probability to be selected. It is achieved by the following steps:

1. Define a ranking function for each chromosome
$$eval(A_i) = \alpha(1 - \alpha)^{(i-1)}, i = 1, 2, ..., pop\_size$$
where $\alpha \in (0, 1)$ is a predefined parameter.
2. Based on this ranking function, calculate the cumulative probability, q_i for each chromosome $A_i \cdot q_i$ is given by
$$q_0 = 0, q_i = \sum_{k=1}^{i} eval(A_k), i = 1, 2, ...pop\_size$$
3. Generate a random real number γ in $(0, q_{pop\_size}]$.
4. Select the i^{th} chromosome A_i such that $q_{i-1} < \gamma \leq q_i$. Repeat step 3 and step 4 until *pop_size* copies of chromosomes are obtained. These *pop_size* copies of selected chromosomes $\{\hat{A}_1, \hat{A}_2, ..., \hat{A}_{pop\_size}\}$ are the mother chromosomes for the reproduction of the next generation.

Crossover: The crossover process will produce a new generation of population based on the set of mother chromosomes $\{\hat{A}_1, \hat{A}_2, ..., \hat{A}_{pop\_size}\}$ resulting from the selection process. We define $P_c \in [0, 1]$ as the probability of crossover. Hence, the expected number of mother chromosomes that will undergo crossover is $P_c \cdot pop\_size$. To pick the parents for crossover, we perform the following action:

For $i = 1$ to *pop_size*
  generate a random number γ;
  if $\gamma < P_c$ put $\hat{A}_i$ in a parent list
  else put $\hat{A}_i$ in a non-parent list
end.

We denote the parent list as $\{\tilde{A}_1, \tilde{A}_2, ...\}$ and the non-parent list as $\{\bar{A}_1, \bar{A}_2, ...\}$. If there are odd number of members in the parent list, the last member will be switch to the non-parent list. With an even number of member in the parent list, we group the members into pairs $\{(\tilde{A}_1, \tilde{A}_2), (\tilde{A}_3, \tilde{A}_4), ...\}$. A random natural number $c \in [2, n]$ is generated and applied to the crossover operation to each parent pair to produce two children given by:

$$X = (\tilde{g}_1^1, ..., \tilde{g}_1^{c-1}, \tilde{g}_2^c, ..., \tilde{g}_2^n)$$
$$Y = (\tilde{g}_2^1, ..., \tilde{g}_2^{c-1}, \tilde{g}_1^c, ..., \tilde{g}_1^n)$$

A new generation of population is produced by combining the children produced by the parent pairs and the non-parent chromosomes.

Mutation: In GA, to avoid the solution being bounded by a local optimum, a mutation process is applied to the chromosomes in the new generation. We define $P_m \in (0, 1)$ as the probability of mutation. Hence, the expected number of chromosomes that will undergo mutation is $P_m \cdot pop\_size$. Similar to the picking of chromosomes for crossover, chromosomes are picked for mutation based on P_m. For each chromosome picked, denoted by $A_l = (g_l^1, g_l^2, ...g_l^n)$, two mutation position n_1 and n_2 is randomly chosen where $1 \leq n_1 < n_2 \leq n - 1$. For $j = n_1$ to n_2, change g_l^j to a random patch from the input example which does not equal to any of the existing genes. After all the genes in and between the two mutation positions are changed, A_l changes to form a new mutated chromosome A_l'.

Termination: Two termination criteria are used. Either the process is executed to produce a fixed number N_g of generations, or no further improvement for the best solution is observed in N_o consecutive generations. The best solution among all these generations (the chromosome with the highest fitness) is chosen as the result when GA is terminated. Otherwise, the algorithm goes back to the evaluation step and begins the next iteration.

Figure 4: Results for image texture synthesis. For each texture, the input is on the left and the output is on the right. All results are generated in real-time with the corresponding ω-tiles.

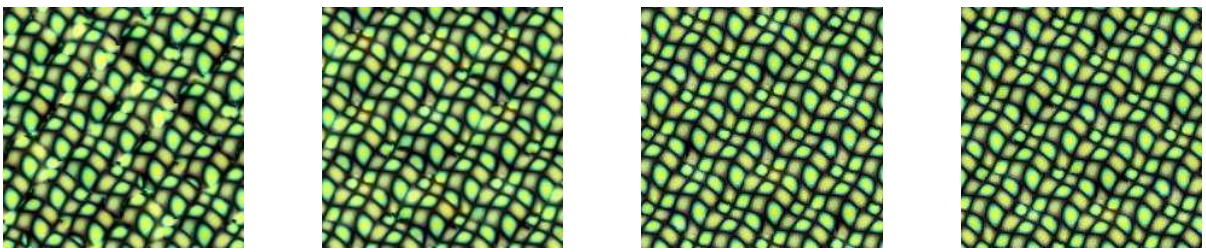


Figure 5: Results comparison of using different population sizes in GA. From left to right: $pop\_size = 10, 20, 40, 80$.

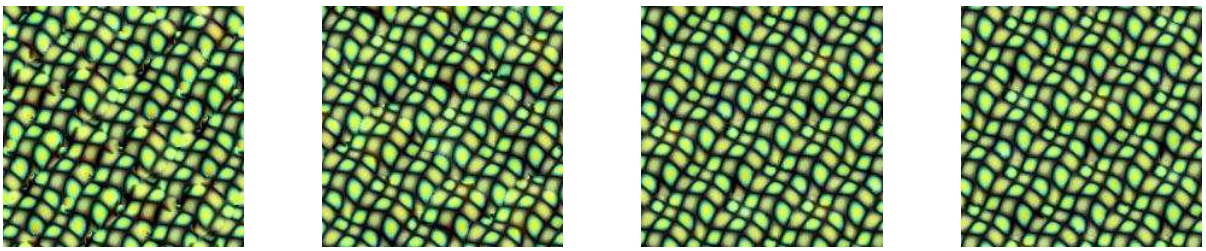


Figure 6: Results comparison of using different generation numbers in GA. From left to right: $N_g = 10, 30, 50, 80$.

4.3 Working Flow

The GA-enhanced sample patches selection algorithm can effectively reduce the boundary artifacts caused by the mergence of the intermediate tile and its matching patch during the graph-cut process. The whole working flow of our optimized tile-based texture synthesis algorithm is: in pre-computation, first randomly initialize a considerable number of sample patches sets (as the chromosomes) from the input example, then use GA to find the best one. Finally graph-cut is employed for junction elimination. The run-time tiling process is the same as [21], we can synthesize arbitrary size of texture images in real-time.

5 Results and Discussion

Figure 4 shows some texture synthesis results using our algorithm. All the results are tiled in real-time with the ω-tile set $\mathbb{N}_2$ which contains 38 tiles (illustrated in Figure 2). Here we use the same parameters for GA with $\{pop\_size = 40, \alpha = 0.05, P_c = 0.3, P_m = 0.2, N_g = 50, N_o = 5\}$ in all the experiments of Figure 4. The most time-consuming procedure in our GA is the fitness evaluation of the chromosomes. In this procedure, we use approximate-nearest-neighbor search (ANN) [1] to accelerate the neighborhood matching operation between each intermediate tile and the input example. Note that the other techniques such as TSVQ [9, 28], FFT [11, 24] and mixture trees [5] can also be employed here. Execution times of the GA-based sample patches selection process are listed in Table 1. All timing results are reported for our unoptimized C++ code on a Pentium 4 3.2GHz PC with 1GB RAM. The sequence of the example names is consistent with the image positions in Figure 4, from left to right and top to bottom. The timings indicate that using GA is an efficient way to select feasible sample patches.

Table 1: Sample patches selection timings for the examples in Figure 4.

Example	Example size	Tile size	GA
Beans	128×128	80×80	25 sec.
Yellow leaves	128×128	80×80	34 sec.
Tree barks	128×128	80×80	28 sec.
Caustics	128×128	80×80	24 sec.
Stones	128×128	80×80	35 sec.
Bread	108×99	70×70	18 sec.
Grape	144×144	100×100	43 sec.
Grass	128×128	80×80	26 sec.
Fabric	128×128	80×80	24 sec.
Bricks	128×128	80×80	37 sec.
Eggs	128×102	80×80	23 sec.

The most important parameters in GA are the population size pop_size and the generation number N_g. In Figure 5 and Figure 6 we show comparisons of using different population sizes and different generation numbers in GA. The other parameters are set to be the same as Figure 4. The input examples are the same as the input texture in the first row of Figure 8. Normally the setting of $\{pop\_size = 40, N_g = 50\}$ is enough for most synthesis.

We can generate the distance function defined in Equation (1) to incorporate other characteristics of the texture besides color. For example, in order to use image gradients as an additional similarity metric, we could define the distance as

$$C(t) = \sum_{p \in S_t} \frac{|T(p) - I(p-t)|^2}{\mu(\|\nabla T\| + \|\nabla I\|)} \quad (2)$$

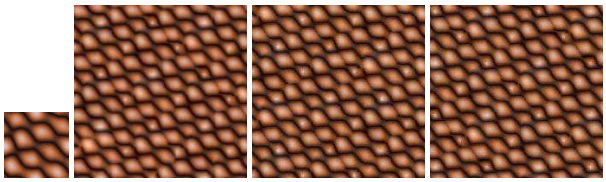


Figure 7: Results comparison when using different distance metrics. From left to right: the input example, result using Equation (1), (2) and (3).

or

$$C(t) = \sum_{p \in S_t} |T(p) - I(p-t)|^2 + \mu \sum_{p \in S_t} |\nabla T - \nabla I|^2 \quad (3)$$

where $\nabla = [\frac{\partial .}{\partial x}, \frac{\partial .}{\partial y}]$ is the gradient operator and μ is a relative weighting coefficient ($\mu = 10$ in our experiments). Figure 7 shows the synthesis results using different distance metrics. Even though we have experimented with color and gradient, one could use other distance metric which measures some property of the texture patch. For most textures, we can simply use the color as the distance metric, as all the experiments in Figure 4 do.

We compare our results with other techniques in Figure 8. We can see that the qualities of our results are comparable with the off-line graph-cut method [14] (even though, their results outperform ours when synthesizing some structural textures) while better than the other CPU-based real-time techniques. The results in Figure 4 and Figure 8 show that our method is a very good choice for textures without very clear structures, especially for natural textures [2].

6 Conclusion and Future Work

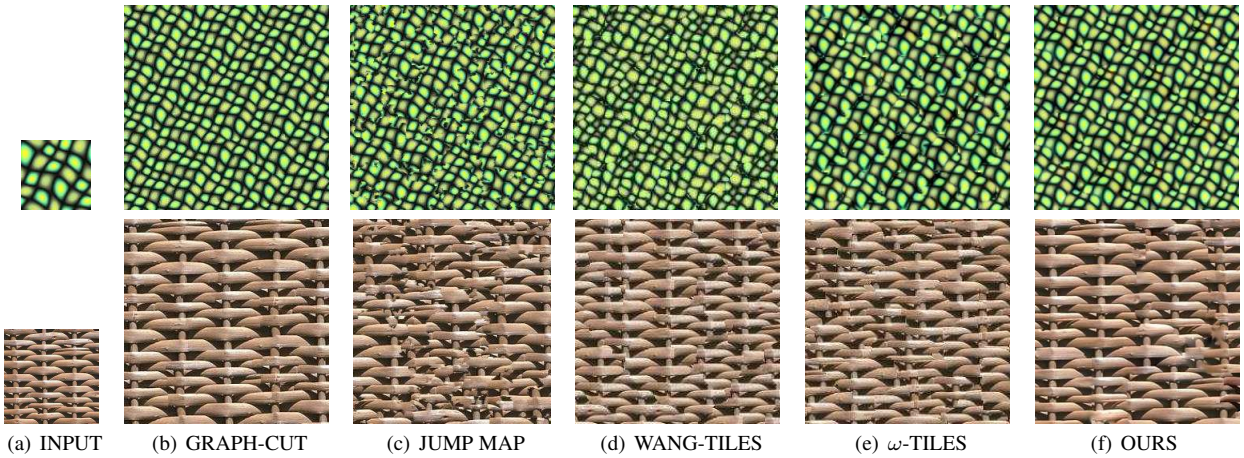
We have presented a novel optimization-based technique for tile-based texture synthesis. Our results for both texture synthesis and image tiling are comparable to state-of-the-arts. We define a pattern repetitive principle that allows us to derive new ω-tile sets from the existing one. An optimized sample patches selection algorithm based on GA is used to improve the quality of the whole tile set. This framework is also fit for quality improvement of Wang-tile based texture synthesis [4]. Our technique can be nicely applied in the environment where real-time texture synthesis is needed, such as 3D games and real-time virtual reality systems, while the local region-growing methods such as image quilting, graph-cut and texture optimization are not applicable (need seconds or minutes to generate an image).

A limitation of our technique is that because it tries to erase the junctions in the intermediate tiles by a single patch from the input example, it is always constrained by the patterns of the intermediate tiles. It is manifested as relatively low qualities when synthesizing some structural textures, for example the eggs texture in Figure 4.

For future work, we wish to extend our tile-based synthesis technique to handle image or geometric textures on 3D models. Another potential direction is to experiment with other local region-growing texture synthesis methods, such as texture optimization [13] and fractional Fourier texture masks [22], in the tile construction step to improve the synthesizing quality of structural textures.

Acknowledgements

We would like to thank anonymous reviewers for all their helpful comments. We thank Steve Zelinka, Vivek Kwatra and Tuen-Young Ng for sharing their results and texture samples on the web sites.



(a) INPUT (b) GRAPH-CUT (c) JUMP MAP (d) WANG-TILES (e) ω-TILES (f) OURS

Figure 8: Comparison of texture synthesis results with various other techniques. Results for other techniques are obtained from their web pages.

REFERENCES

[1] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[2] Michael Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.

[3] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[4] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.

[5] Frank Dellaert, Vivek Kwatra, and Sang Min Oh. Mixture trees for modeling and fast conditional sampling with applications in vision and graphics. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 619–624, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Weiming Dong, Shuangxian Sun, and Jean-Claude Paul. Optimal sample patches selection for tile-based texture synthesis. In *CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 503–508, Washington, DC, USA, 2005. IEEE Computer Society.

[7] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[8] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1033, Washington, DC, USA, 1999. IEEE Computer Society.

[9] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[10] John H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[11] Steven L. Kilthau, Mark S. Drew, and Torsten Möller. Full search content independent block matching based on the fast fourier transform. In *Proceedings of International Conference on Image Processing 2002*, volume 1, pages 669–672, Vancouver, BC, Canada, 2002.

[12] John R. Koza. Survey of genetic algorithms and genetic programming. In *Proceedings of 1995 WESCON Conference*, pages 589–594. IEEE, 1995.

[13] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24(3):795–802, 2005.

[14] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003.

[15] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, 2005.

[16] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. *ACM Trans. Graph.*, 25(3):541–548, 2006.

[17] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[18] Boading Liu and Broading Liu. *Theory and Practice of Uncertain Programming*. Physica-Verlag, 2002.

[19] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near-regular texture analysis and manipulation. *ACM Trans. Graph.*, 23(3):368–376, 2004.

[20] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[21] Tuen-Young Ng, Conghua Wen, Tiow-Seng Tan, Xinyu Zhang, and Young J. Kim. Generating an ω-tile set for texture synthesis. In *Proceedings of Computer Graphics International 2005 (CGI'05)*, pages 177–184, Stone Brook, NY, USA, 2005.

[22] A. Nicoll, J. Meseth, G. Müller, and R. Klein. Fractional fourier texture masks: Guiding near-regular texture synthesis. *Computer Graphics Forum*, 24(3):569–579, September 2005.

[23] J.S. Pan, F.R. McInnes, and M.A. Jack. Application of parallel genetic algorithm and property of multipleglobal optima to vq codevector index assignment for noisy channels. *Electronics Letters*, 32(4):296–297, 1996.

[24] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. *ACM Trans. Graph.*, 21(3):673–680, 2002.

[25] M. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[26] Hongwei Sun, Kwok-Yan Lam, Siu-Leung Chung, Weiming Dong, Ming Gu, and Jiaguang Sun. Efficient vector quantization using genetic algorithm. *Neural Comput. Appl.*, 14(3):203–211, 2005.

[27] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 55–63, New York, NY, USA, 2004. ACM Press.

[28] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-

Wesley Publishing Co.

[29] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM Trans. Graph.*, 23(3):364–367, 2004.

[30] Steve Zelinka and Michael Garland. Towards real-time texture synthesis with the jump map. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 99–104, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[31] Steve Zelinka and Michael Garland. Jump map-based interactive texture synthesis. *ACM Trans. Graph.*, 23(4):930–962, 2004.